57PRTS

09/831240
JC18 Rec'd PCT/PTO 03 MAY 2001

SHADING AND TEXTURING 3-DIMENSIONAL COMPUTER GENERATED IMAGES

This invention relates to a method and apparatus for shading and texturing 3-dimensional computer generated images.

The standard method used when generating 3-dimensional images for shading and texturing is the z or depth buffer system. This has been in use for many years and is now an industry standard.

In our British Patent No. 2281682, we have proposed an alternative method with many advantages over the z buffer system. It is a ray casting technique in which each object (polygon) in an image is defined by a set of infinite surfaces which are tagged as forward or reverse facing. A ray from a viewpoint is cast into the scene and intersections with the surfaces are determined along with the distance of the intersections from an image plane. By analysing this data, it is possible to determine which surface of each polygon is visible for each pixel and thus it is possible to shade and texture the pixel. The technique may be implemented by using a pipeline of processing elements, each of which can perform a ray surface interception calculation to determine the depth value for that surface for each pixel in turn. When the depth value for a surface at a pixel has been determined, using these processing elements it is stored in a depth or z buffer with data identifying the surface. The z buffer information can then be read out and used by a texturing device to apply texture data for display via a frame store.

Surfaces in the system can be opaque or they may have translucency. Blending values can be associated with the surfaces to enable translucency effects to be displayed. For example, a cloud may be modelled as a translucent surface by assigning the RGB values to define its colour and an alpha blending value to define its translucency. The degree of translucency is controlled by modulating the alpha value of the texture map across the surface. A commonly used blending mode is known as alpha blending where successive polygons are blended into the frame buffer and put into the equation:

RGB(new) = alpha*RGB(frame buffer) + (1-alpha)*RGB (texture)

This technique is well known. A restriction when using alpha blending is, that in order to render a correctly blended image, the pixels which display the polygons must be presented to the blending unit in depth order rendered from back to front so that each contribute the correct amount for the final image. For a z buffer system, this ordering is typically performed by the application software that is feeding the z buffer. This is referred to as pre-sort mode. The ray casting technique can use either the pre-sort mode or an automatic pixel accurate type of sort.

There is a special case of translucent texture referred to as "punch through". This is defined as a translucent texture where the alpha component is restricted to be either "on", i.e., fully transparent, or "off", i.e., fully opaque. This type of texture is very common in 3-D game applications for two reasons; firstly it allows complex scenes like forests to be modelled using relatively few polygons; and, secondly, a traditional z buffer can correctly render punch through translucency irrespective of the order in which polygons are presented to the system.

A traditional z buffer pipeline with alpha testing for translucency is shown in Figure 1. In this, polygons to be rendered are first scan line converted in the polygon scan converter 2 and the resulting x,y,z,u,v,w values are then fed to the texturing unit and to the depth test unit (the z buffer). Texture addresses are sent to texture memory. Texture values retrieved from texture memory pass to a texture filter 4 which reduces aliasing artifacts introduced by the texture resampling process. The filtered values are passed to a texture blend unit 6 which blends the texture values with the base colour and highlights of the polygon. Next, an alpha test is performed on the alpha component of the resulting pixel in an alpha test unit 8. The test is against a reference alpha value.

The alpha test unit performs a magnitude comparison with an alpha reference value. The user supplies the alpha reference value, and a compare mode which is one of "never", "less", "equal", "less or equal", "greater", "not equal", "greater or equal", or "always". The test selected depends on the type of image the user is creating. The alpha test unit outputs whether or not the compare mode has been passed or failed by the input alpha value.

If the result of the alpha test is a pass then the depth test is performed on the z value and RGB values in the z buffer updated. If the test is failed, then no further processing of pixels takes place. Thus, in the typical case where the reference alpha is one and the test is greater than or equal, then only opaque pixels are written to the frame buffer, and these pixels are depth tested against all other polygons in the scene, irrespective of the order in which polygons are processed. The depth test is performed by a depth test unit 10 which reads z values from the z buffer 12 via a stencil unit 14 and is able to write the z values directly to the z buffer store. The depth test conditions are shown in Figure 1 next to the depth test unit.

Figure 2 shows the ray casting technique of our British Patent No. 2281662 with deferred texturing, that is to say, texturing data is applied at the end of the ray/surface intersection testing process. The apparatus comprises a polygon setup unit which provides data which defines the plane equations of the polygons. This data is then fed to an array of processing elements 22, each of which is able to perform one ray/surface intersection test and can produce a depth value for that ray/surface intersection. Typically, the processing array operates on a sub-set of the whole image known as a "tile" which is e.g., 32 x 32 pixels and operates on one polygon at a time. The depth values computed are stored in a tile depth store 24.

Depth and surface values from the tile depth store are applied to a parallel runlength encoder 26. This then provides an output to the texturing unit via an XY address for a pixel which is fed to a polygon scan conversion unit 28 and a tag which goes to the polygon setup unit 30. This calculates the plane equations for the texturing and shading, and recalculates the plane equations of the polygons.

This setup unit receives the polygon texture data and then provides data to the polygon scan conversion unit 28. The RGB values and alpha values and highlights applied to a surface are then sent to a texture blending unit 32 where they are combined with RGB and alpha values from texture memory via a filter 34 and are then passed to the tile accumulation buffer from where they will be sent to the main frame store for the image.

The advantage of deferring texturing until all the ray/surface intersections have been performed is that texture and shading operations then only occur on visible pixels. Thus, effective fill rate is increased by the ratio of occluded to visible pixels in the scene. By definition, in a deferred texturing pipeline the depth test occurs before the texturing takes place and, therefore, also takes place before the alpha test. As a result, it is not possible to correctly render the punch through textured polygons unless they are either presented in back to front order and are non-overlapping, or they are subject to a pixel accurate pre-sorting operation earlier in the pipeline. The first of these arrangements imposes an unnecessary restriction in the software driver. The second requires an unnecessary processing overhead.

We have appreciated that by including a feedback loop to the depth store from the results of the alpha test for each pixel and deferring the updating of the depth store until the results of the alpha test for each pixel are known this problem may be overcome. This requires both feedback from the alpha testing to the depth testing and feedforward from the depth test unit to the alpha blending unit.

Using this idea enables four standard modes of operation to be used as follows:

1. A standard z buffer no-sort, punch through compatibility mode;

2. An enhanced punch through mode with no-sort and deferred operation;

3. An enhanced punch through mode with auto-sorting surfaces and deferred operation;

4. Generalised auto-sort alpha blend mode.

These modes of operation and preferred embodiments of the invention will now be described in detail by way of example with reference to the accompanying drawings in which:

Figure 1 is the prior art z buffer system described above;

Figure 2 is the prior art ray casting system with deferred texturing;

Figure 3 is an embodiment of the present invention;

Figure 4 is a further preferred embodiment of the invention with an auto-sort unit;

Figure 5 shows schematically the auto-sort scheme for polygons; and

Figure 6 shows schematically a further auto-sort scheme for polygons.

The block diagram of Figure 3 is a modification of that of Figure 2 in accordance with an embodiment of the invention.

As in Figure 2, a polygon setup block 20 receives polygon vertex data before supplying polygon surface data to the processor element array 22 which computes the depths of each surface at each pixel and stores them. This also includes a depth tester to determine the closest surface to the image plane for each pixel. This can supply the standard ABC and tag parameters which are used to define each surface to a bypass FIFO store 40. This in turn can supply those parameters back to the processor element array 22 when controlled so to do.

XY address data and tag data is supplied by the parallel runlength encoder 26 to the polygon setup unit 30 and the polygon scan converter 28 in a similar manner to Figure 2. The polygon setup unit receives polygon texture data from the texture store. The blending unit 32 and filter 34 operate in a similar manner to that of Figure 2. However, after the texture blending unit, an alpha test unit 42 is provided. This has a pass/fail output which is used as a control to a punch through control block 44 which also receives an alpha value and position data from the alpha test unit. The alpha test unit has an output to a conditional blend unit 46 which can blend the textured data directly with data from the tile depth store 24 before supplying the data to the tile accumulation buffer 36.

There is also an auto sort unit, for surface data, connected to the tile depth store. This sends data to a vertex requester 50 which then ensures that polygon vertex data is supplied in an appropriate order to the polygon setup unit 20.

It has a sort direction control so that it can sort surfaces from front to back and from back to front and also a bypass control to inhibit its use. Use of the circuitry will be described in four different modes.

The z buffer is such a widely used rendering algorithm that it is useful to have a mode of operation which emulates this since this is schematically what many programmers will be familiar with. Thus, the advantages of the ray casting rendering algorithm are obtained while appearing to be operating in a z buffer mode.

In this mode, polygons bypass the auto sort logic 48 as a result of a control signal sent to its bypass input. Instead, they pass to the depth test unit associated with tile depth store 24 unsorted. Rather than perform the depth calculation at this point, the standard ABC and tag parameters which form the basis of the ray casting technique are stored in the bypass FIFO 40 and the screen XY locations and tag values are forwarded to the deferred texturing unit for each surface in turn.

In this mode of operation texturing is not deferred, as all pixels are presented to the texture unit regardless of whether they have been occluded by a previously rendered opaque pixel which is closer to the eyepoint in the current pixel. The texture unit processes "texels" (texture elements) in the order in which they are presented. Texels are filtered, blended with the polygon base colour, and any highlights, and are alpha tested in the same way as for a traditional z buffer (see Figure 1). For each pixel, if the alpha test is passed, the corresponding depth is calculated from the stored ABC and tag parameters in FIFO 40 which are fed back to the processor element array 22 and then into the tile depth store 24. The corresponding depth is then calculated with the associated depth test unit and the tile depth store is updated. The pixel RGB value is then blended into the tile accumulation buffer by the conditional alpha blend unit 46. This process continues until all the polygons in the scene have been processed. The speed of operation of this is equivalent to a standard z buffer since all the polygons have to be textured and shaded. There are no economies to be made by only texturing and shading the visible polygons.

The second mode of operation is one in which punch through control is used to speed up processing but there is no sorting of polygons and surface data prior to shading and texturing. The mode is the same as that described for z buffer operation with the exception that the polygons are depth tested against the current contents of the tile depth store before being passed to the texturing unit. Punch through textures are detected by the alpha test unit 42 which sends signals to punch through control unit 44 to send the punch through surfaces back to the processor element array 22. They are then sent to the tile depth store and depth test unit 26 which tests them against the current contents of the store. If they are closer to the image plane then they replace the current contents. If a succeeding polygon or a part of a polygon is occluded at a particular pixel by a previously rendered opaque pixel, then it is not sent to the texturing unit. Thus, the degree of fill rate improvement over the z buffer system will depend on two factors; firstly, the order in which the polygons are presented; and, secondly, the latency between the depth test and the alpha test. For example, if polygons happen to be presented in a back to front order and the polygons are non-overlapping, the fill rate will be identical to the z buffer system. If, on the other hand, the polygons happen to be presented in front to back order, in a 0 latency system, the fill rate requirement will be reduced by the ratio of ideal occluded pixels in the scene to visible pixels in the scene, (i.e., the depth complexity). The effect of latency in the system is to reduce the efficiency of this process, since for a finite time window (the degree of latency in the system), a proportion of pixels which could have been identified as occluded in a 0 latency system will be passed to texturing unit and will consume texturing bandwidth. This latency is inherently because there is a time period which will be taken to texture the first polygon to pass through the texturing unit and alpha and depth test it. Once this initial latency has been overcome, the processor element arrangement and the tile depth store and depth test unit 24 will be provided with depths for new polygons and performing depth tests on previously rendered polygons when the result of the alpha test is that the pixel in question is a punch through pixel occluding other pixels in the scene.

A third mode of operation is a deferred texturing pipeline process which uses a translucency sorting algorithm.

As described above, the degree of fill rate improvement provided by the invention depends on the order in which the polygons are presented. The advantage of using punch through control in combination with pixel accurate auto sorting is such that a system can ensure that polygons and parts of polygons are always processed in optimal order. The auto sort algorithm for punch through processing differs from the algorithm for alpha blend processing in two key respects. Firstly, the punch through pixels are sorted from front to back rather than back to front. This is because if there is a punch through pixel, it will have an alpha value of 1 and no surfaces behind that will contribute to the shade and texture applied to that pixel. Secondly, the sorting overhead can be reduced by decomposing the sort list into smaller chunks. We shall refer to this as "chunking".

We shall describe the operation of punch through auto sort algorithms with reference to Figure 4. This shows the portion of Figures 2 and 3 which performs the ray casting algorithm but does not show the texturing hardware.

As can be seen, a cache memory 52 is provided between the polygon setup unit 20 and the processing element array 22. The tile depth store and depth test unit 24 has an output to Auto Sort logic 54 which is coupled to auto discard logic 58 and chunking logic 54. The chunking logic has an output back to the cache 52 and also to vertex requester unit 50. The Auto Sort logic 56 comprises at least one additional tile depth store for temporary storage of data. When looking for opaque surfaces these are not required as a comparison with the currently stored depth determines whether or not an opaque pixel is closer than the current depth. They are used when sorting translucent objects. These additional depth stores could be provided in the tile depth store and depth test unit 24.

Because punch through textures are typically used to model complex structures such as forests and cloud scenes, the number of polygons used for punch through is generally much greater than that used for alpha blended translucency. Consequently, the processing overhead for sorting has to be minimised to make the system efficient. A simple algorithm to sort n object into a oscending or descending order would require n^2 operations. This is illustrated in Figure 5 in which a tree punch through texture can be seen on the front surface, which has opaque texels in the region of the tree and transparent ones around it. Auto sorting is performed by rendering all of the translucent polygons in order to recognise the bottom layer. Therefore, the number of polygons per tile processed is the number of polygons multiplied by the number of layers which is $2n^2$.

If each object selected is discarded from the sorting operation, the number of operations is

$$n+n-1 + n-2 + \ldots\ldots +1$$

which is approximately equal to $n^2/2$. The characteristic of punch through sorting is that the sort can be chunked without introducing any visual errors. For chunking, the number of operations is $MOD(n/m)*m^2+REM(n/m)^2$, where n is the chunk size. Reducing the chunk size reduces both the sorting overhead and the efficiency of the sort. Any real implementation of this algorithm will be a trade-off between the computing resource available used in the sort algorithm and the efficiency of the punch through operation. The benefit of chunking is that it increases the degree of freedom within which this trade-off can be explored. The sorting algorithm can be further optimised unless it can terminate early, at the point at which all the pixels in a tile have become validated (i.e., they have been assigned a valid opaque depth). This is illustrated with reference to Figure 6. In tile (a) the nearest polygon hides all polygons behind it. Therefore, the number of polygons processed to n.

A single pass through the punch through sorting algorithm will find the closest polygon to the image plane for each pixel. Then, in the case of tile (a), the first pass will determine that a closest opaque polygon has been found for each pixel in the tile, and therefore no further passes are necessary.

Chunks of polygon data are supplied to the tile depth store, depth test logic and auto sort logic by the chunking logic 54. They then provide into the discard logic polygons which are no longer required.

The auto sort logic then sorts the polygons to be rendered in a direction controlled by a sort direction input and supplies them to cache memory 52 from where they can then be sent again to the processor element array 22 and eventually on through to the texturing unit. Thus, the auto sort logic sorts the polygons into front to back order and when used in the apparatus of Figure 3, is able to significantly reduce the processing overhead in complex scenes, since the closest fully opaque pixels will be processed first and polygons behind these will not need to be textured.

The fourth mode of operation of the circuit of Figure 3 is a generalised auto sort alpha blend mode.

With state of the art graphic controllers performing bi-linear and tri-linear texturing as standard operations, the blocky visual nature of punch through textures is becoming less acceptable. Typical art work, for say, a tree texture would be punch through (i.e., fully opaque) in the body of the tree and alpha blended (i.e., partially translucent) of the extremities of the leaves. It is possible with the circuit of Figure 3 to correctly render this type of alpha blended texture whilst still retaining the benefit of deferred texturing.

In order to do this, two passes through the tree surface data would be made. In the first pass, the punch through portion of each polygon in a tile is processed as described in the auto sort mode above. The alpha test unit 42 is set to pass only fully opaque texels, so that at the end of the pass the tile depth buffer contains the depth of the closest fully opaque texel to the eye for each pixel position in the tile. In the second pass, the auto sort algorithm sorts the pixels from back to front, and the depth test is set to "greater than" so that for each visible polygon fraction, only the non-opaque (i.e., the alpha blended fringe of the leaves in the tree example) is passed to the texturing unit. Because the second pass sorts back to front, any overlapping partially translucent textures will be correctly blended. The second pass should be very efficient because in a typical scene, (i.e., a punch through forest) only a small percentage of the total pixels in the polygon list will pass the depth test. The suitability of this technique for general alpha blended textures depends on the proportion of opaque texels to translucent texels in the texture maps. In the tree example, the algorithm would be most suitable. If, however, all the textures were clouds and contained no opaque pixels, then standard auto sorting would be the preferred method.

It will therefore be appreciated from the above that methods and apparatus which operate according to the ray casting technique for rendering 3-dimensional images can be modified to obtain the benefit of reduced processing overheads from punch through textures, thus speeding up the operation of the system. The use of the punch through textures is particularly beneficial in scenes where there are a large number of overlapping punch through textures such as cloud scenes or forests. The use of the punch through processing means that only the polygon closest to the viewpoint for each pixel has to have a texture value applied to it. Without this operation it would be necessary to apply textures to all the polygons which intercept the ray which passes through that pixel. In a scene such as a forest, this could easily lead to several thousand texturing operations being required for each pixel instead of one. Thus, very complex scenes can be textured at much higher speeds than would otherwise be possible.